

|  |  |
| --- | --- |
| **Comments** | |
| **Operator** | **Operation / Example** |
| # | *Inline comment*; *#* text here |
| """ """ | *Function docstring*; Def example ():  *"""*text here *"""* |
|  |  |

|  |  |  |
| --- | --- | --- |
| **Math Operators** | | |
| **Operators** | **Operation** | **Equivalent / Example** |
| \*\* | Exponent. | 2 \*\* 3 # 8 |
| % | Modulus/Remainder. | 22 % 8 # 6 |
| // | Integer division. | 22 // 8 # 2 |
| / | Division. | 22 / 8 # 2.75 |
| \* | Multiplication. | 3 \* 3 # 9 |
| ‘text’ \* 2  # “textext” |
| - | Subtraction. | 5 - 2 # 3 |
| + | Addition. | 2 + 2 # 4 |

|  |  |  |
| --- | --- | --- |
| **Walrus Operator** | | |
| **Operator** | **Operation** | **Example** |
| := | Assign variable within return expression. | print(var:="Hello")  # “Hello” |



|  |  |
| --- | --- |
| **len() function** | |
| **Keyword** | **Operation/ Example** |
| len() | *Returns length of string, number of values in list/dictionary/tuple*;  *len*(“text”) # 4  *len*([“hi”, 2, “tada”]) # 3 |

|  |  |
| --- | --- |
| **Input() function** | |
| **Keyword** | **Operation/ Example** |
| Input()  /  Input(“default”) | *Take input from user*;  print(“what is your name?”)  my\_name = *input()* /  my\_name = *input*(“what is your name?”) |

|  |  |  |
| --- | --- | --- |
| **Augmented Assignment Operators** | | |
| **Operators** | **Equivalent** | **Example** |
| var += 1 | var = var + 1 | greeting = “Hello”  greeting += “ world!”  # “Hello world!” |
| var -= 1 | var = var - 1 |
| var \*= 1 | var = var \* 1 |
| var /= 1 | var = var / 1 | number = 1  number += 1 # 2 |
| var //= 1 | var = var // 1 |
| var %= 1 | var = var % 1 | my\_list = [“item”]  my\_list \*= 3  # [“item”, “item”, “item”] |
| var \*\*= 1 | var = var \*\* 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Data Types** | | | | | |
| **Type** | **Example(s)** | **Operation** | **Type** | **Example(s)** | **Operation** |
| Integer | -2, 0,1 | int(“4”) # 4 | set | {1,2,3} | set([1,2,3]) #{1,2,3} |
| Float | -1.25, 0.0, 1.5 | float(“4.5”) # 4.5 | bool | 0 | bool(0) # False |
| String | “hello” | str(4) # “4” | byte | b"hello" | bytes("hello", "utf8") |
| List | [1, 2, 33] | List((1, 2)) # [1, 2] | byte array | bytearray(b"ob") | bytearray("ro", "utf8")  # bytearray(b"ob") |
| Tuple | (3, 4, 44) | Tuple([1, 2]) # (1,2) |
| Dictionary | {“a”: 1, “b”:2} | dict(<variable>) |  |  |  |

|  |  |
| --- | --- |
| **Print function** | |
| **Keyword** | **Operation/ Example** |
| print() | print(“text”) # “text” |
| end= | *Avoid newline after print*;  a = [“text”, “here”]  for word in a:  print(word, *end*=“-”)  #” text-here ” |
| sep= | *Separator of vars in print*;  print(“text”, “here”,*sep*= “,” )  # “text,here” |



|  |  |  |
| --- | --- | --- |
| **Comparison Operators** | | |
| **Operators** | **Operation** | **Equivalent / Example** |
| == | Equal to | “a” == “a” # True |
| != | Not equal to | “a” != “b” # True |
| < | Less then | 1 < 2 # True |
| > | Greater than | 2 > 1 # True |
| <= | Less or equal to | 1 <= 1 # True |
| >= | Greater or equal to | 2 >= 2 # True |

|  |  |
| --- | --- |
| **Boolean Operators** | |
| **Operators** | **Operation/ Example** |
| and | True and Trues # True  True and False # False |
| or | True or False # True  False or False # False |
| not | not True # False  not False # True |



|  |  |
| --- | --- |
| **Def function** | |
| To run a piece of code external or repeatly | |
| def | *Def name(keyword, keyword):* execute the code below using the keywords |
| keyword | *Def name (keyword = variable):* can use both the order or use = to indicate |
| return | *return variable* returns the value of the value where it was called for |
| global | Global variable  Variable = value after global is called the next time variable is set it is then a global value |
| def plusone( keyword):  global old\_value  oldvalue = keyword  return keyword + 1 | |

|  |  |
| --- | --- |
| **If statement** | |
| if | *If Boolean query is true:* execute next line of code |
| elif | If first if Boolean query == False, if *elif Boolean query == true:* execute next line of code (optional, unlimited) |
| else | If all if and elif Boolean queries return False*: else:* execute next line of code (optional) |
| Age == 27  If age < 18:  print(“kid”)  elif age > 40:  print(“grandpa”)  else:  print(“middle aged”) | |

|  |  |
| --- | --- |
| **Switch-case statement** | |
| Like an if statement but just for == | |
| match | *match variable*: start the switch case and select a single variable |
| case | *Case value:* check of the value == the match variable if so execute next line of code |
| or | *Case value or value:* makes it possible to check if either values == to the match variable if so execute next line of code |
| [] / [\*rest] | *Case [a]:* Makes it possible to react differently on different amount of values in a iteritable  *Case [a, b, \*rest]:*  \*rest indicating the else (default) |
| Int(), str(), float() | *case int():* lets it possible to react differently on different classes and activating the next line of code |
| \_ | *Case \_:* lets you add an else (default) |
| code = 300  match code:  case 69:  print(“nice”)  case 420 or 8008:  print(“funny number”)  case str():  print(“well that”s not a number”)  case [a , b]:  print(“it”s a list of 2 codes”)  case [a, b, \*rest]:  print(“a list of more than 2 codes”)  case \_:  print(“this is the default”) | |



|  |  |
| --- | --- |
| **While loop statement** | |
| while | *While if statement:* Repeat line below till statement is true |
| break | *If statement:*  *break* forces too exit loop |
| continue | *If statement:*  *continue* forces to return to start of loop without running code below, thus skips the rest |
| number = 0  while number < 10:  number += 1  if number == 3:  continue # Skips printing 3  if number == 8:  break # Exits the loop completely  print(number) | |



|  |
| --- |
| **Template Strings** |
| Library which allows to create reusable templates for the creation of strings using variables. |
| from string import Template  name = "robin"  t = Template("hey $name!")  t.substitute(name=name)  # hey robin |

|  |  |
| --- | --- |
| **Install a library** | |
| **Command** | **Description** |
| py -m pip install <library> | Installs an python package aka a library on the operating pc. |

|  |  |
| --- | --- |
| **Pretty print** | |
| pprint.pprint(variable) splits the data object in more readable print | |
| import pprint | |
| list | listz = [1,2]  pprint.pprint(listz)  #1  # 2 |
| dict | dictz = {"a": 1, "b":2}  pprint.pprint(dictz)  # { "a": 1,  # "b", 2 } |

|  |  |
| --- | --- |
| **lambda function** | |
| A single line anonymous function | |
| lambda | Function\_name = *lambda* keyword, keyword: actual function |
| add = lambda x, y: x + y  add(5, 3) # 8 | |

|  |  |
| --- | --- |
| **Sys.exit() function** | |
| **Keyword** | **Operation/ Example** |
| exit() | *Import sys*  *Sys.exit()* closes python when ran |

|  |  |
| --- | --- |
| **Range() function** | |
| **Keyword** | **Operation/ Example** |
| range() | *range(3)* creates a sequence of numbers starting with 0 # (0, 1, 2) |
| *range(0, 6, 2)* if given multiple arguments it will be read as start, stop, increment # (0, 2, 4) |
| *range(3, -1, -1)* if given a – it will count down # (3, 2, 1, 0) |

|  |  |
| --- | --- |
| **For loop statement** | |
| For | *for x in iteratable:* run next line of code that many times |
| else | *else:* run when for loop has been completed without hitting a *break* |
| numbers = [1, 4, 2, 5, 7]  target\_number = 5  for number in numbers:  if number == target\_number:  print("Found the target number!")  break  else:  print("Target number not found in the list.") | |



|  |  |
| --- | --- |
| **Working with tuples** | |
| + more memory efficient than list,  + ordered  + duplicates allowed  **-** not iterable, changeable  - less performance than sets | |
| **what** | **How**  Eg: tuple = (“a”, “b” ) |
| Getting value by index | tuple [0] # “a”  tuple [-1] # “b” |
| getting sub tuple with slices (creating a copy of the og tuple) | tuple [:] # (“a”, “b”)  tuple [:2]  # [“a”, “b”]  tuple [1:] # ( “b” ) |
| in  not in | “a” in tuple # True  “z” not in tuple  # True |
| getting a index() | tuple.index(“a”) #0 |

|  |  |
| --- | --- |
| **Working with lists** | |
| + iterable, changeable, variable data types in one  + ordered  + duplicates allowed  - Less memory efficient than tuples | |
| **what** | **How**  Eg: list = [“a”, “b” ,”c” ,”d”] |
| Getting value by index | list[0] # “a”  list[-1] # “d” |
| getting sub lists with slices (creating a copy of the og list) | list [:] # [“a”, “b” ,”c” ,”d”]  list [1:3] # [ “b” ,”c” ]  list [1:-1] # [“b” ,”c” ,”d”]  list [:2] # [“a”, “b”]  list [1:] # [ “b” ,”c” ,”d”] |
| replacing a value | list[0] = “z” # [“z”, “b” ,”c” ,”d”] |
| concatenation | list = list + [1 ,2]  # [“a”, “b” ,”c” ,”d”, 1 2] |
| in  not in | “a” in [“a”, “b” ,”c” ,”d”] # True  “z” not in [“a”, “b” ,”c” ,”d”] # True |
| assignment trick | var1, var2, var3, var4 = list  # “a” # “b” # “c” #”d” |
| getting a index() | list.index(“a”) #0 |
| add value at the end append() | list.append(“z”)  # [“a”, “b” ,”c” ,”d”, “z”] |
| add value on specified place insert() | list.insert(1, “z”)  # [“a”, “z”, “b” ,”c” ,”d”] |
| remove value using index or slice del() | del list[0] # [“b” ,”c” ,”d”]  del list[1:3] # [“a”, “d”] |
| remove first instance of a value using a value remove() | list.remove(“a”) # [ “b” ,”c” ,”d”] |
| dele last pop() | list.pop() # [“a”, “b” ,”c”] |
| sorting a list sort(reverse=True) | list.sort() # [“a”, “b” ,”c” ,”d”]  list.sort(reverse=True)  # [“d”, “c” ,”b” ,”a”] |



|  |  |
| --- | --- |
| **List comprehension** | |
| **Syntax** | **Description** |
| new\_list = [i for i in oldlist] | Creates a new list from an old list, accept conditionals too. |

|  |  |
| --- | --- |
| **Working with sets** | |
| + No duplicates allowed  + Support math operations  + Good performance  **-** unordered (cant be indexed)  - Limited modification | |
| **what** | **How**  e.g. set= {1,2,3} |
| Removes duplicates. | s= {1,2,2,3}  # {1,2,3 |
| Add to a single value to a set. | set.add(4)  #{1,2,3,4} |
| Adding multiple values to a set. | set.update({3,4,5])  #{1,2,3,4,5} |
| Removing a value. | set.discard(3)  #{1,2} |
| union sets | s2 = {3,4}  set.union(s2) #{1,2,3,4} |
| Return only the common values. | set.intersection(s2)  #{3} |
| The one sets unique values. | set.difference(s2)  #{1,2} |
| Return only unique values. | set.symmetric\_difference(s2) #{1,2,4} |



|  |  |
| --- | --- |
| **Working with Dictionaries** | |
| **what** | **How**  Eg: dict = {“a”: “z”, “b”:2, “c”:3} |
| getting a translation using [] (error when non existent) | dict[“b”]  # 2 |
| getting a translation using .get() (“none” or a given default when non existent | dict.get(“a”) # “z”  dict.get(“g”) # “None”  dict.get(“g”, “default”)  # “default” |
| adding a pair using [] | dict[“d”] = 4  #{“a”: “z”, “b”:2, “c”:3, “d”:4} |
| adding items (only when non existent with setdefault() | list.setdefault("d", True)  #{“a”: “z”, “b”:2, “c”:3, "d", True} |
| get a list of the values .values() | dict.values()  # [“z”, 2, 3] |
| get a list of the keys .keys() | dict.keys()  #[“a”, “b”, “c”] |
| get a list of the items  .items() | dict.items()  #[(“a”: “z”), (“b”:2), (“c”:3)] |
| remove an item using the key. pop()  and returns the value | dict.pop("a") # "z"  print(dict)  # { “b”:2, “c”:3} |
| remove the last added item popitem() and returns it as a tuple | dict.popitem()  # (“c”, 3)  print(dict)  # {“a”: “z”, “b”:2} |
| remove an item using the key del() | dict.del("a")  # { “b”:2, “c”:3} |
| remove all items in the dictionary clear() | dict.clear()  # {} |
| check if key is in the dictonary | "a" in dict.keys()  # True |
| check if value in the dictonary | 3 in dict.values()  # True |
| merge dictionaries | dict2 = {"d":4}  dict3 = {\*\*dict, \*\*dict2}  # {“a”: “z”, “b”:2, “c”:3, "d":4} |

|  |  |  |
| --- | --- | --- |
| **Formatting Floats** | | |
| **For mat** | **Res ult** | **Description** |
| {:.2f} | 3.14 | Format float 2 decimal places |
| {:+.2f} | +3.14 | Format float 2 decimal places with sign |
| {:+.2f} | -1.00 | Format float 2 decimal places with sign |
| {:.0f} | 3 | Format float with no decimal places |
| {:0>2d} | 04 | Pad number with zeros (left padding, width 2) |
| {:x<4d} | 4xxx | Pad number with x’s (right padding, width 4) |
| {:x<4d} | 10xx | Pad number with x’s (right padding, width 4) |
| {:,} | 1,000,000 | Number format with comma separator |
| {:.2%} | 35.00% | Format percentage |
| {:.2e} | 1.00e+09 | Exponent notation |
| {:11d} | 11 | Right-aligned (default, width 10) |
| {:<11d} | 11 | Left-aligned (width 10) |
| {:^11d} | 11 | Center aligned (width 10) |



|  |
| --- |
| **Writing appending files** |
| with open('bacon.txt', 'w') as file:  #With "w" indicating write, or "a" can be used to append instead.  file.write('Hello world!\n') |

|  |
| --- |
| **Opening and reading files** |
| with open('C:\\Users\\hi.txt') as file: #open file  file\_content = file.read() # Reads files as one big string """ """.  file.readlines() #Reads all lines with each line of the file being an individual record of the list. |



|  |  |
| --- | --- |
| **Regular expressions aka Regex 1** | |
| Regex is a sequence of characters that specify a search pattern. In python you use it using import re. | |
| **Symbol** | **Matches** |
| ? | zero or one of the preceding group. |
| \* | zero or more of the preceding group. |
| + | one or more of the preceding group. |
| {n} | exactly n of the preceding group. |
| {n,} | n or more of the preceding group. |
| {,m} | 0 to m of the preceding group. |
| {n,m} | at least n and at most m of the preceding p. |
| {n,m}? or \*? or +? | performs a non-greedy match of the preceding p. |
| ^spam | means the string must begin with spam. |
| spam$ | means the string must end with spam. |
| . | any character, except newline characters. |
| \d, \w, and \s | a digit, word, or space character, respectively. |
| \D, \W, and \S | anything except a digit, word, or space, respectively. |
| [abc] | any character between the brackets (such as a, b, ). |
| [^abc] | any character that isn’t between the bracket |
| **Basic use of regex** | |
| # Compile a pattern  phone\_num\_regex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')  # Search the string.  mo = phone\_num\_regex.search('My number is 415-555-4242.')  mo.group() #"415-555-4242" | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Working with strings 1** | | | | | |
| **Escape characters** | | | | | |
| **Character** | **Prints as** | | **Character** | | **Prints as** |
| \' | Single quote ' | | \" | | Double quote " |
| \t | Tab | | \n | | Newline |
| \\ | Backslash \ | | \b | | Backspace |
| \ooo | Ooctal value | | \r | | Carriage return |
| **String formatting** | | | | | |
| **Format** | | **Description** | | **Example** | |
| Raw string / r string | | Ignores escaped characters. | | s = r"/pizza/" # /pizza/ | |
| %s (string) %d (int / decimal) | | Replace % with variable. | | name = "robin"  "hello %s" % name #"hello robin" | |
| .format(<variable>, <variable>) | | Replaces {} with variable | | surname= "goldenberg"  "Hello I am {} {}".format(name, surname) #"hello I am robin Goldenberg" | |
| f string | | Allows variables being placed inbetween the {} if an = is placed behind the variable it will use both the experession and its value. | | "hello I am {name=} {surname}" #"hello I am name= robin Goldenberg" | |



|  |  |
| --- | --- |
| **Working with strings 2 Operations** | |
| **What** | **How** e.g. s = "pizza Yum" |
| Indexing | s[0] #"p" |
| Slicing | s[0:5] #"pizza" |
| in / not in operator.  Returns true when string is in string. | "pizza" in s #True |
| upper() capitalize all characters. | s.upper() #"PIZZA YUM" |
| lower() lower all characters. | s.lower() #"pizza yum" |
| title() capitalize first letters | s.title() #"Pizza Yum" |
| isupper/ islower() operators  Returns true if string is all uppers. | "HELLO".isupper() #True |
| isalpha() Returns true if strings consist only of letters. | "HELLO".isalpha() #True |
| isalnum() returns true if string only consists of letters and numbers. | "Helo124".isalnum() #True |
| isdecimal() returns true if string consist only of numbers. | "123".isdecimal() #True |
| isspace() returns true if string only consists of space, tabs, new lines. | " ".isspace() #True |
| istitle() returns true if strings only consists of words that begin with an uppercase and followed by only lowercase characters. | "Pizza Yum".istitle() #True |
| Startswith() / endswith().  Returns true a string starts with another string. | s.startswith("pizza") #True |
| Join() joins an entire list/dictionary/tuple/set into a string. | "".join(["pizza", "yum"]) #"pizza yum" |
| Split(<string>) splits a string on the given string by default it splits on whitespace. | s.split() #["pizza", "yum"]  s.split("a") #["pizz", " yum"] |
| rjust(<int>, <string>) / ljust(<int>) / center(<int>).  pads the string on the left, right or both sides too the given int, if no string is given it uses the space character. | "hello".rjust(10, "-") #"-----Hello"  "hello".center(10) #" Hello " |
| strip(<string>)/ restrip()/ lstrip()  strips any occurrence of the given string. If no string is given it will remove whitespaces. | s.strip() #"pizzayum"  s.strip(" yum") #"pizza" |
| .count(<string>) returns int representing the count of times the given string occurred inside the string . | s.count("a") #1 |
| .replace(<string>, <string>) replaces the first string with the second string inside the original string. | s.replace("Yum", "bleg") #pizza bleg |



|  |  |
| --- | --- |
| **Regex 2 Own Character class** | |
| **What** | **How** |
| Using [ ] you can create own character call, returning a list of each of the occurrence of the letters found in the list. | vowel\_regex = re.compile(r'[aeiouAEIOU]')  vowel\_regex.findall('Robocop eats baby food. BABY FOOD.')  ## ['o', 'o', 'o', 'e', 'a', 'a', 'o', 'o', 'A', 'O', 'O'] |
| using ^ at the beginning of the list results in an opposite working. | consonant\_regex = re.compile(r'[^aeiouAEIOU]')  consonant\_regex.findall('Robocop eats baby food. BABY FOOD.') # ['R', 'b', 'c', 'p', ' ', 't', 's', ' ', 'b', 'b', 'y', ' ', 'f', 'd', '.', '', 'B', 'B', 'Y', ' ', 'F', 'D', '.'] |



|  |  |
| --- | --- |
| **Regex 3 advanced** | |
| **What** | **How** |
| Multiple groups. Using () | phone\_num\_regex = re.compile(r'(\d\d\d)-(\d\d\d-\d\d\d\d)')  mo = phone\_num\_regex.search('My number is 415-555-4242.')  mo.group(1) #"415" group() still returns all |
| Multiple patters. Using | | hero\_regex = re.compile (r'Batman|Tina Fey')  mo1 = hero\_regex.search('Batman and Tina Fey.')  mo1.group() #"batman" |
| Optional groups using ? | bat\_regex = re.compile(r'Bat(wo)?man')  mo1 = bat\_regex.search('The Adventures of Batman')  mo1.group() #"batman" |
| Allowing multiple of the same. Using + | bat\_regex = re.compile(r'Bat(wo)+man')  mo2 = bat\_regex.search('The Adventures of Batwowowowoman')  mo2.group() #"Batwowowowoman" returns None of the group isn’t found at all. |
| optional grouping but allowing multiple of the same. Using \* | bat\_regex = re.compile(r'Bat(wo)\*man')  mo3 = bat\_regex.search('The Adventures of Batwowowowoman')  mo3.group()#"Batwowowowoman" Also allows the group to be non existing. |
| Using ^ to indicate that a match should start with the pattern. With $ indicating it should end with the pattern. | begins\_with\_hello = re.compile(r'^Hello')  begins\_with\_hello.search('Hello world!')  # <\_sre.SRE\_Match object; span=(0, 5), match='Hello'> |
| Wildcard character using .  with .\* having any length of wildcard characters, except newline chars. | at\_regex = re.compile(r'.at')  at\_regex.findall('The cat in the hat sat on the flat mat.')  #['cat', 'hat', 'sat', 'lat', 'mat'] |
| by adding re.IGNORECASE too the patter it will ignore case letters. | robocop = re.compile(r'robocop', re.IGNORECASE)  robocop.search('Robocop is part man, part machine, all cop.').group()  # 'Robocop' |



|  |  |
| --- | --- |
| **Regex 5 sub()** | |
| **Description** | **Example** |
| Instead of finding causes sub to replace the pattern with a separate given string. | names\_regex = re.compile(r'Agent \w+')  names\_regex.sub('CENSORED', 'Agent Alice gave the secret documents to Agent Bob.') #'CENSORED gave the secret documents to CENSORED.' |

|  |  |
| --- | --- |
| **Regex 4 findall()** | |
| **Description** | **Example** |
| Findall() returns a list of all occurrences of the pattern. | phone\_num\_regex = re.compile(r'\d\d\d-\d\d\d-\d\d\d\d')  phone\_num\_regex.findall('Cell: 415-555-9999 Work: 212-555-0000') #['415-555-9999', '212-555-0000'] |



|  |  |  |
| --- | --- | --- |
| **File path** | | |
| **What** | **How Os (windows)**  import os | **How pathlib ( linux/ mac)**  from pathlib import Path |
| Creating a file path | os.path.join("usr","bin", "spam")  # "usr\\bin\\spam" | Path("usr").joinpath("bin") .joinpath("spam"))  # usr/bin/spam |
| Retrieve current working directory. | os.getcwd() #C:\\Python34 | Path.cwd() #C:\\Python34 |
| Change current working directory. | os.chdir("c:\\windows\\system") |  |
| Create new directory. | os.makedirs("C:\\pizza") | (Path.cwd / "pizza").mkdir() |
| Checking if file/directory exists. | os.path.exists(<path>) | Path(<path>).exists() |
| Checking if path is a file. | os.path.isfile(<path>) | Path(<path>).is\_file() |
| Check if path is a directory | os.path.isdir(<path>) | Path(<path>).is\_dir() |
| Get file size | os.path.getsize(<path>) | Path(<path>).stat() |



|  |  |
| --- | --- |
| **Shutil copy/move files** | |
| **What** | **How**  Import shutil |
| Copy a file. | shutil.copy(<filepath>, <target filepath>) |
| Copy a directory. | shutil.copy(<directory path>, <target directorypath>) |
| Move and rename. | shutil.copy(<filepath>, <target filepath + new name>) |

|  |  |
| --- | --- |
| **Delete files/ directory** | |
| **What** | **How**  Import os, shutil |
| Delete file | os.unlink(<path + file>) |
| Delete empty directory | os.rmdir(<path>) |
| Delete an directory and all its content | shutil.rmtree(<path>) |

|  |  |
| --- | --- |
| **JSON writing/ reading** | |
| To read and write json files you can use the json library.  While the reading is similar to reading any other file, writing uses the dump function. | |
| dump() accepts 3 arguments: content being a dictonary, f being the json file and ident indicating the indication inside the json the dictionary should be written too. | |
| **Reading a json**  import json | **Writing a json, dictonary 🡪 json**  import json |
| import json  with open("filename.json", "r") as f:  content = json.load(f) | content = {"name": "Joe", "age": 20}  with open("filename.json", "w") as f:  json.dump(content, f, indent=2) |



|  |  |
| --- | --- |
| **Exception handling** | |
| **Try:** | |
| Indicates the start of the exception block | |
| **Except <error type> as <variable>:** | |
| Providing an except is mandatory when using Try:. You can chain multiple except blocks to provide different messages, or execute different lines of code when certain errors occurs. The error message will be saved in the <variable>. Except can be used to prevent a script from crashing from user error. | |
| **Errortype examples (there are many more)** | |
| Exception | Catches most error types and provides useful info in the variable. |
| BaseException | Catches all errors but only provides limited information in the variable. |
| ZeroDivision Error | catches zero division errors. |
| **traceback.format\_exc()** | |
| Can be imported from the traceback library. It provides not only the error message but also the line the error occurred on. It adds this information to an variable. | |
| **traceback.print\_exc()** | |
| It’s the same as format\_exc() but instead print the error details. | |
| **Finally:** | |
| everything under the finally: block gets executed no matter if the try: block was successful or hit an exception. Useful for closing database connections. | |
| **Example** | |
| def divide(dividend , divisor):  try:  print(dividend / divisor)  except ZeroDivisionError:  print('You can not divide by 0')  except BaseException:  traceback.print\_exc()  finally:  print('Execution finished') | |

|  |
| --- |
| **YAML** |
| To read a yaml file (json but with comments) the ruamel library can be used.  first you have to create a YAML() constructor object and than you can load data into it. |
| **Example** |
| from ruamel.yaml import YAML  with open("filename.yaml") as f:  yaml=YAML()  yaml.load(f) |



|  |
| --- |
| **Raise** |
| Raise is often used in combination with an if statement as when it is executed it will raise the specified custom error.  This error can be used in combination with try, except. |
| **Example** |
| try:  raise MyCustomException('A custom  message for my custom exception') except MyCustomException:  print('My custom exception was raised') |
| Another method to use raise is by combining it with Exception and an exception message. There by raising an error and stoping the script from continuing to run. |
| **Example** |
| if len(symbol) != 1:  raise Exception('Symbol must be a single  character string.') |

|  |
| --- |
| **JSON YAML ect 🡪 dictionary** |
| To read a json, yaml ect. Files into a python dictonary the anyconfig library can be used |
| **Example** |
| import anyconfig  conf1 = anyconfig.load("/path/to/foo/conf.d/a.yml") |



|  |  |  |
| --- | --- | --- |
| **Logging** | | |
| The logging library can be used to keep a log of what when is happening in the script and if wanted can be written to a file.  There are two methods to set up logging with or without the specification of a log file.  you have to set up logging for each level you intend to use. | | |
| **Level** | **Function** | **Description** |
| DEBUG | logging. debug() | The lowest level. Used for small details. Usually you care about these messages only when diagnosing problems. |
| INFO | logging. info() | Used to record information on general events in your program or confirm that things are working at their point in the program. |
| WARNING | logging. warning() | Used to indicate a potential problem that doesn’t prevent the program from working but might do so in the future. |
| ERROR | logging. error() | Used to record an error that caused the program to fail to do something. |
| CRITICAL | logging. critical() | The highest level. Used to indicate a fatal error that has caused or is about to cause the program to stop running entirely. |
| **Setting up example without log file** | | |
| import logging  logging.basicConfig(level=logging.DEBUG, format=' %(asctime)s - %(levelname)s- %(message)s') | | |
| **Setting up example with log file** | | |
| import logging  logging.basicConfig(filename='myProgramLog.txt', level=logging.DEBUG, format='%(asctime)s - %(levelname)s - %(message)s') | | |
| **Example of log writing** | | |
| logging.debug("start of program")  # 2015-05-23 16:20:12,664 - DEBUG - Start of program | | |

|  |
| --- |
| **Assert** |
| Assert is used to do sanity checks on variables. If the sanity check is failed it will raise an error.  to create an assertion you start with assert followed by the variable and condition you want to check and then separated by a "," the error message may it fail. |
| **Example** |
| door = 'open'  assert door == 'open', 'The door needs to be "open".' |



|  |
| --- |
| **Args and Kwargs** |
| To indicate an undefined amount of arguments to a function \*args can be used. Or \*\*kwargs for dictonaries. |
| **Example** |
| def some\_function(\*args, \*\*kwargs):  ….code  some\_function('arg1', 'arg2', 'arg3')  # Arguments passed: ('arg1', 'arg2', 'arg3') as <class 'tuple'>  some\_function(key1='arg1', key2='arg2')  # keywords: {'key1': 'arg1', 'key2': 'arg2'} as <class 'dict'> |

|  |
| --- |
| **Disabling logging** |
| Once you don’t need to logging information any more (say when script is finished and bug free). You can disable the logging without having too delete it everywhere which could cause accidental mistakes. You have to do this for each level separately. By using logging.disable() . |
| **Example** |
| logging.basicConfig(level=logging.INFO, format=' %(asctime)s -%(levelname)s - %(message)s')  logging.disable(logging.CRITICAL) |



|  |
| --- |
| **Inheritance** |
| Inheritance of classes can be done through using it during original class in the creation of a sub class. |
| **Example** |
| class Animal:  def \_\_init\_\_(self, name):  self.name = name  def speak(self):  print("")  class Cat(Animal):  def speak(self):  print("Meow!")  cat = Cat("Whiskers"  cat.speak() |

|  |
| --- |
| **Abstraction** |
| Abstract classes are only meant to be used as a template. |
| **Example** |
| from abc import ABC, abstractmethod  class Shape(ABC):  @abstractmethod  def area(self):  pass  class Rectangle(Shape):  def \_\_init\_\_(self, width, height):  self.width = width  self.height = height |

|  |
| --- |
| **Encapsulation** |
| While python does not have a build in private, protected, public it is generally accepted to use \_ for protected variables and \_\_for private |
| **Example** |
| class MyClass:  def \_\_init\_\_(self):  self.\_protected\_var = 10  self.\_\_private\_var = 20 |

|  |
| --- |
| **With statement** |
| The with statement is used as context manager, when a context ends the object is closed automatically. |
| **Example** |
| with open(filename) as f:  file\_contents = f.read() |
| **Custom context manager** |
| Its also possible to write a custom context manager using context lib. |
| **Example** |
| import contextlib  @contextlib.contextmanager  def context\_manager(num):  print('Enter')  yield num + 1  print('Exit')  with context\_manager(2) as cm:  # the following instructions are run when  # the 'yield' point of the context manager is  # reached. 'cm' will have the value that was yielded  print('Right in the middle with cm = {}'.format(cm))  # Enter  # Right in the middle with cm = 3  # Exit |

|  |
| --- |
| **Decorators** |
| A decorator is a function which takes another function as an argument.  by adding @functoos.wraps python keeps the original function without changes made inside the wrapper for debugging purposes. |
| **Example** |
| import functools  def your\_decorator(func):  @functools.wraps(func) # For preserving the metadata of func.  def wrapper(\*args,\*\*kwargs):  # ….. Do stuff before func  result = func(\*args,\*\*kwargs)  # …. Do stuff after func  return result  return wrapper |



|  |
| --- |
| **Dataclass** |
| To create your own data type @dataclass can be used. By using the = you can give it a default value.  By giving it the data type Any it accepts any. |
| **Example** |
| from dataclasses import dataclass  from typing import Any  @dataclass  class Number:  val: int  count: int = 0  value: Any  obj = Number(2)  obj.val  # 2  obj.count  # 0 |



|  |
| --- |
| **ABS()** |
| Return the absolute value of a number. If the argument is a complex number its magnitude is returned. |
| **Example** |
| abs(-1)  #1  abs(3.14)  #3.14  abs(0x10)  #16  abs(3 + 2j)  # 3.6055512754639896  abs(0o20)  #16 |

|  |
| --- |
| **Poetry** |
| Poetry is a tool for dependency management and packaging in Python. It allows you to declare the libraries your project depends on and it will manage (install/update) them for you. |
| **Example** |
| py -m pip install --user poetry  #Create new project.  poetry new my-project  #Add dependencies to the project. pyproject.toml file  [tool.poetry.dependencies]  pendulum = "^1.4"  #Adding it through cmd.  $ poetry add pendulum  #Install the dependencies.  poetry install  # Remove dependencies.  poetry remove pendulum |

|  |
| --- |
| **\_\_main\_\_** |
| By using if\_\_name\_\_ == "\_\_main\_\_": everything under it is only ran when it is ran as a standalone execution rather than when its being called upon. |
| **Example** |
| if \_\_name\_\_ == "\_\_main\_\_":  pass |

|  |
| --- |
| **Setup.py** |
| It is normal to create a setup.py for each project, specifying the python version, licence and all the installed packages.  using py <path>/setup.py install, all the needed libraries can than be installed, before running. |
| **Example** |
| from distutils.core import setup  setup(  name='pythonCheatsheet',  version='0.1',  packages=['pipenv',],  license='MIT',  long\_description=open('README.txt').read(),  ) |



|  |
| --- |
| **Virtual Environment** |
| The use of a Virtual Environment is to test python code in encapsulated environments, and to also avoid filling the base Python installation with libraries we might use for only one project. |
| **Example virtual env** |
| pip install virtualenv  pip install virtualenvwrapper-win  #Make an virtual environment.  mkvirtualenv <environment name>  #Bind the env too current working directory.  setprojectdir .  #Deactive the environment  deactivate  #Workon a project.  workon <environment name> |
| **Example Anaconda** |
| #Create env.  conda create -n <environment name>  # Activate env.  conda activate <environment name>  # Exit env.  conda deactivate |
| **Example pipenv** |
| pip install pipenv  #Enter your Project directory and install the Packages for your project.  cd <project name>  pipenv install <package>  #Uninstall Packages.  pipenv uninstall <package>  #Activate the Virtual Environment associated with your Python project.  pipenv shell  #Exit the Virtual Environment.  exit |



|  |
| --- |
| **All()** |
| Return True if all element of the iterable are true. |
| **Example** |
| all([True, 1 , True])  # True  all((0, True, False))  # False |

|  |
| --- |
| **Aiter()** |
| Return an asynchronous iterator for an asynchronous iterable. Its like a build in threading. |
| **Example** |
| async def aitersync(iterable):  results = []  async for x in aiter(iterable):  results.append(x)  return iter(results) |



|  |
| --- |
| **Chr() / Ord()** |
| Chr() returns the string representing a character. Accepts an int representing a Unicode character |
| **Example** |
| print(chr(97))  #"a" |
| ord() does the exact opposite of chr() takes an Unicode character and returns the corresponding integer. |
| **Example** |
| print(ord"a")  # 97 |

|  |
| --- |
| **Callable()** |
| Return True if object is callable. |
| **Example** |
| x = 5  print(callable(x))  # False  def function():  pass  y = function  print(callable(y)  # True  z lambda: None  print(callable(z)  # True |

|  |
| --- |
| **Breakpoint()** |
| Opens the python debugger at the moment breakpoint() is reached. |
| **Example** |
| if i == 3:  breakpoint() |

|  |
| --- |
| **ascii()** |
| Return a string with a printable representation of an object. But escape non-ascii characters. |
| **Example** |
| ascii('A')  # 'A'  ascii('ë')  # '\xeb'  ascii(['A', 'ë'])  # ['A', '\xeb'] |

|  |
| --- |
| **Bin()** |
| Convert an integer number to a binary string prefixed with "0b". Accepts an int. |
| **Example** |
| bin(1)  # '0b1'  bin(10)  # '0b1010 |

|  |
| --- |
| **Any()** |
| Return True if any element of the iterable is true. |
| **Example** |
| any([False, 0, 0])  #False  any ((0, True, False))  # True |



|  |
| --- |
| **Compile() Exec() Eval()** |
| compile() Compile the source (string) into a code or abstract syntax tree (AST) object. |
| exec() supports dynamic execution of python code. |
| **Examples** |
| code\_string = """  def hello\_world():  print('Hello, world!')  """  # Compile the string into a code object  code\_object = compile(code\_string, '<string>', 'exec')  # Execute the code object  exec(code\_object)  # Call the function defined in the code object  hello\_world()  #hello world |
| code\_to\_execute = "print('Hello, exec()!')"  exec(code\_to\_execute  #Hello, exec()! |
| eval() Evaluates and executes an expression. |
| **Example** |
| eval('1 + 4')  # 5  eval('print("Hello World!")')  # Hello World! |

|  |
| --- |
| **Classmethod()** |
| Transform a method into a class method. |
| **Example** |
| class Counter:  def \_\_init\_\_(self):  self.count = 0  def increment(self):  self.count += 1  return self.count  @classmethod # passes the class as the first argument to the method instead of passing the instance  def get\_new\_instance(cls):  return cls()  if \_\_name\_\_ == "\_\_main\_\_":  counter = Counter()  print(counter.increment())  # 1  counter = counter.get\_new\_instance()  print(counter.increment())  # 1 |



|  |
| --- |
| **Delattr()** |
| Delets the named attribute, provided the object allows it. E.g. deleting the age of person class object. |
| **Example** |
| class Person:  name = "John"  age = 36  country = "Norway"  delattr(Person, 'age') |

|  |
| --- |
| **Dir()** |
| Returns a list of all available attributes of an object, module, class. .pyfile |
| **Example** |
| a = 10  print(dir() #['\_\_annotations\_\_', '\_\_builtins\_\_', '\_\_doc\_\_', '\_\_loader\_\_', '\_\_name\_\_', '\_\_package\_\_', '\_\_spec\_\_', 'a']  class MyClass:  def \_\_init\_\_(self):  self.x = 5  obj = MyClass()  print(dir(obj)) # Output: ['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', ... 'x', ] |

|  |
| --- |
| **Complex()** |
| Return a complex number with the value real + imag\*1j |
| **Example** |
| complex(1)  # (1+0j)  complex('1')  # (1+0j) |



|  |
| --- |
| **Enumerate()** |
| Return an enumerate object. A sequence of an iterable object containing a sort of dict for values and their indexes. |
| **Example** |
| for i, item in enumerate([1, 2, 3, 4, 5]):  print(f"Index: {i}, Item: {item}")  # Index: 0, Item: 1  # Index: 1, Item: 2  # Index: 2, Item: 3  # Index: 3, Item: 4  # Index: 4, Item: 5 |

|  |
| --- |
| **Divmod()** |
| Return a pair of numbers consisting of their quotient and remainder. |
| **Example** |
| divmod(2, 2)  # (1, 0)  divmod(10, 2)  # (5, 0)  divmod(7, 2)  # (3, 1) |



|  |
| --- |
| **Hash()** |
| Transform an immutable object into an int. |
| **Example** |
| hashed\_string = hash("robin") #43491451697 |

|  |
| --- |
| **Hasattr()** |
| Returns true if the given string is the same as the name of an objects attribute. |
| **Example** |
| class Example:  attribute = "Hello, World!"  obj = Example()  print(hasattr(obj, "attribute") #True |

|  |
| --- |
| **Globals()** |
| Returns all global variables within the current scope of the python code returns it a dictonary. |
| **Example** |
| print(globals()  # {a: "value"} |

|  |
| --- |
| **Getattr()** |
| Return the value of the named attribute of an object. It accepts 3 arguments, the object, the attribute you want to retrieve and the value that needs to be returned if the attribute doesn’t exist. |
| **Example** |
| class Example:  attribute = "Hello, World!"  obj = Example()  value = getattr(obj, 'attribute', 'Nothing found')  print(value) #Hello, World! |

|  |
| --- |
| **Frozenset()** |
| Return a new frozenset object. It takes a list or tuple ect and creates a set which is immutable no things can be added or removed from it |
| **Example** |
| frozenset([1, 2, 3])  # {1, 2,3} |

|  |
| --- |
| **Filter() / Map()** |
| Combining lambda with a filter function you can filter only those values that fulfil the lambda function. |
| **Example** |
| numbers = [1, 2, 3, 4, 5, 6]  even\_numbers = list(filter(lambda x: x % 2 == 0, numbers))  print(even\_numbers) #[2, 4, 6] |
| Combinding lambda with a map function result in the lambda function being applied to each of the values of a list |
| **Example** |
| numbers = [1, 2, 3, 4]  squared\_numbers = list(map(lambda x: x\*\*2, numbers))  print(squared\_numbers) #[1, 4, 9, 16] |



|  |
| --- |
| **Isinstance()** |
| Return True if the object argument is an instance of an object. This can be checking if a variable is of a certain type or if it is a certain class object. |
| **Example** |
| isinstance(1, int) # True  isinstance(1, str) # False  class Dog(Animal):  pass  my\_pet = Dog()  is\_dog = isinstance(my\_pet, Dog) #True |

|  |
| --- |
| **Help()** |
| Invoke the built-in help system, returns some info of the applied function. |
| **Example** |
| help(help)  # Help on \_Helper in module \_sitebuiltins object: … |

|  |
| --- |
| **Hex()** |
| Convert an integer number to a lowercase hexadecimal string. Prefixed with "0x". |
| **Example** |
| hex(1) # '0x1'  hex(10) # '0xa'  hex(100) # '0x64'  hex(1000) # '0x3e8' |



|  |
| --- |
| **Next()** |
| Retrieve the next item from the iterator. |
| **Example** |
| I = *ite*r([1, 2, 3])  i.\_\_next\_\_() #1 |

|  |
| --- |
| **Locals()** |
| Returns a dictionary with all local variables. |
| **Example** |
| def my\_function():  name = "Jim"  age = 35  print(*locals())*  my\_function() # {'name': 'Jim', 'age': 35} |

|  |
| --- |
| **Min()** |
| Return the smallest item in an iterable. |
| **Example** |
| *min*([1, 2, 10, 40, 5]) # 1 |

|  |
| --- |
| **Issubclass()** |
| Return True if class is a subclass of classinfo. |
| **Example** |
| class First:  pass  class Second(First):  pass  print(issubclass(Second, First)) # True  print(issubclass(First, Second)) # False |

|  |
| --- |
| **Id()** |
| Return the “identity” of an object. This is an integer which is guaranteed to be unique and constant for this object during its lifetime. Two objects with non-overlapping lifetimes may have the same id() value. |
| **Example** |
| id(1) # 9788960 |

|  |
| --- |
| **Max()** |
| Return the largest item in an iterable. |
| **Example** |
| *max*([1, 2, 10, 40, 5]) # 40 |

|  |
| --- |
| **Iter()** |
| Return an iterator object. Making it possible for next to be used. |
| **Example** |
| i = iter([1, 2, 3])  I # <list\_iterator object at 0x7f93158badc0>  i.\_\_next\_\_() # 1  i.\_\_next\_\_() # 2  i.\_\_next\_\_() # 3 |



|  |
| --- |
| **Object()** |
| Creates a new, featureless object. |
| **Example** |
| my\_object = *object()*  print(type(my\_object)) # Output: <class 'object'> |



|  |
| --- |
| **Sorted()** |
| Return a new sorted list from the items in iterable. |
| **Example** |
| *sorted*([1, 2, 3, 7, 4] ) # [1, 2, 3, 4, 7]  *sorted*(['a', 'h', 'e']) # ['a', 'e', 'h'] |

|  |
| --- |
| **Setattr()** |
| Is used to set the value of an attribute on an object. It takes three arguments:  object: The object you want to modify.  name: The name of the attribute you want to set.  value: The new value for the attribute |
| **Example** |
| class Person:  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  person = Person("Alice", 30)  setattr(person, "age", 31)  print(person.age) # Output: 31 |

|  |
| --- |
| **Round()** |
| Return number rounded to ndigits precision after the decimal point. |
| **Example** |
| round(1.4) # 1  round(1.5) # 2  round(2.9) # 3  round(2/3, ndigits=3) # 0.667 |

|  |
| --- |
| **Reversed()** |
| Return a reverse iterator. |
| **Example** |
| i = [1,2,3] print(*reversed*(i)) #<reversed object at 0x00000177A6240E20>  b = "pizza"  reversed\_b = ''.join(*reversed*(b))  print(reversed\_b) #azzip |

|  |
| --- |
| **Repr()** |
| Return a string containing a printable representation of an object. |
| **Example** |
| class Person:  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  person = Person("Alice", 30)  print(*repr*(person)) # Output: <\_\_main\_\_.Person object at 0x7f89420a66d0> |

|  |
| --- |
| **Property()** |
| Return a property attribute. |
| **Example** |
| property(1)  # <property object at 0x7f93158c8590> |

|  |
| --- |
| **Pow()** |
| Given a string representing one Unicode character, return an integer representing the Unicode code point of that character. |
| **Example** |
| result = *pow*(2, 3)  print(result) # 8 |

|  |
| --- |
| **Oct()** |
| Convert an integer number to an octal string prefixed with “0o”. |
| **Example** |
| *oct*(1) # '0o1'  *oct(*10) # '0o12' |



|  |
| --- |
| **Sum()** |
| Sums start and the items of an iterable from left to right and returns the total. |
| **Example** |
| *sum*([2, 4, 6]) # 12 |

|  |
| --- |
| **@staticmethod** |
| The @staticmethod is a function decorator that will transform a class method into a static method. Statics methods can be changed or accessed just called, use is to create function classes. |
| **Example** |
| class Class:  *@staticmethod*  def function():  print("X")  Class.function()  # X |



|  |
| --- |
| **Zip()** |
| Iterate over several iterables in parallel, producing tuples with an item from each one. |
| **Example** |
| furniture = ['table', 'chair', 'rack', 'shelf']  price = [100, 50, 80, 40]  for item, amount in *zip*(furniture, price):  print(f'The {item} costs ${amount}')  # The table costs $100  # The chair costs $50  # The rack costs $80  # The shelf costs $40 |

|  |
| --- |
| **Vars()** |
| Return the \_\_dict\_\_ attribute for a module, class, instance, or any other object with a \_\_dict\_\_ attribute. |
| **Example** |
| class Person:  def \_\_init\_\_(self, name, age):  self.name = name  self.age = age  my\_person = Person("Dwight", 35)  my\_vars = *vars*(my\_person)  print(my\_vars) # {'name': 'Dwight', 'age': 35} |

|  |
| --- |
| **Type()** |
| Return the type of an object. |
| **Example** |
| *type(*'span') # <class 'str'>  *type*(99) # <class 'int'>  *type*(1.1 )# <class 'float'>  *type*([1, 2]) # <class 'list'>  *type*((1, 2)) # <class 'tuple'>  *type*({1, 2}) # <class 'set'>  *type*({'a': 1, 'b': 2}) # <class 'dict'> |

|  |
| --- |
| **Super()** |
| is used to access the parent class of an object. |
| **Example** |
| class Animal:  def \_\_init\_\_(self, name):  self.name = name  def speak(self):  print("Generic animal sound")  class Dog(Animal):  def speak(self):  print("Woof!")  super().speak()  dog = Dog("Buddy")  dog.speak()  #woof!  #Generic animal sound |



|  |
| --- |
| **Json Library** |
| A JSON object is similar to a Python dictionary, but has the following differences:  [  JSON Keys are always string,  Strings are always enclosed with double quotes,  A JSON boolean start with lowercase letters,  null is the JSON equivalent of Python None,  ]  The data types JSON supports are: [String, Number, boolean, null, Object, Array] |
| **JSON loads() method** |
| With the json.loads method, you can parse a JSON object and transform it to a Python dictionary. |
| **Example** |
| import json  json\_person = '{"name": "Charles", "age": 33, "has\_hair": false, "hobbies": ["photography", running"]}'  python\_person *= json.loads*(json\_person)  python\_person  # {'name': 'Charles', 'age': 33, 'has\_hair': False, 'hobbies': ['photography', 'running']}  type(python\_person)  # <class 'dict'>  python\_person.get("name")  # 'Charles' |
| **JSON dumps() method** |
| The other way around. The dumps() method transforms a Python object to a JSON string. |
| **Example** |
| import json  python\_person = {'name': 'Charles', 'age': 33, 'has\_hair': False, 'hobbies': ['photography', 'running']}  json\_person *= json.dumps*(python\_person)  json\_person  # '{"name": "Charles", "age": 33, "has\_hair": false, "hobbies": ["photography", "running"]}'  type(json\_person) # <class 'str'> |
| **Reading a Json File** |
| import json  with open("filename.json", "r") as f:  json\_content = json.loads(f.read())  json.loads(json\_content)  # {'name': 'Charles', 'age': 33, 'has\_hair': False} |
| **Writing a Json File** |
| import json  person = {'name': 'Charles', 'age': 33}  with open("filename.json", "w") as f:  f.write(json.dumps(person)) |



|  |
| --- |
| **Datetime Library 1** |
| The Datetime module allows us to work with date and time objects. It provides three additional data types: date, time and datetime. |
| **Date()** |
| The date method return a date object with the year, month and day attributes:  datetime.date(year: int, month: int, day: int) |
| **Example** |
| *from datetime import date*  obj = *date*(2022, 12, 1)  obj.year # 2022  obj.month # 12  obj.day # 1 |
| **Time()** |
| The time method return a time object with the hour, minute, second, microsecond and tzinfo attributes: datetime.time(hour: int, minute: int, second: int) |
| **Example** |
| *from datetime import time*  obj = *time*(10, 20, 33)  obj.hour # 10  obj.second # 33  obj.microsecond # 0 |
| **Datetime()** |
| The datetime returns an object with both, the date and time objects attributes: datetime.datetime(year, month, day, hour, minute, second) |
| **Example** |
| *from datetime import datetime*  obj = *datetime*(2024, 12, 1, 15, 35, 59)  obj.year # 2024  obj.month # 12  obj.second # 59 |
| **Now() today()** |
| Now and today methods return a datetime object with system’s exact day and time. Additionally, now can take a timezone object as an optional parameter |
| **Example** |
| *from datetime import datetime, timezone*  now = *datetime.now()*  now # datetime.datetime(2022, 7, 23, 19, 56, 49, 589806)  now.date() # datetime.date(2022, 7, 23)  now.time() # datetime.time(19, 56, 49, 589806)  now.year # 2022  now.month # 7  now.second # 49  now.microsecond # 589806  datetime.now(timezone.utc) # datetime.datetime(2022, 7, 24, 0, 20, 8, 265634, tzinfo=datetime.timezone.utc) |



|  |  |  |
| --- | --- | --- |
| **Datetime Library 2** | | |
| **strftime()** | | |
| strftime allow us to create human formatted strings out of a Python datetime object. | | |
| **Example** | | |
| *from datetime import datetime*  now = datetime.now()  now # datetime.datetime(2022, 7, 23, 20, 31, 19, 751479)  now.*strftime*("%d-%b-%Y") # '23-Jul-2022'  now.s*trftime*("%m/%d/%Y") # '07/23/2022'  now.*strftime*("%b/%d/%Y - %H:%M:%S") # 'Jul/23/2022 - 20:31:19' | | |
| **strptime()** | | |
| The strptime method creates a datetime object from a string. obj.strptime(datetime\_string, format) | | |
| **Example** | | |
| *from datetime import datetime*  datetime\_str = '12-Jul-2023'  datetime.*strptime*(datetime\_str, '%d-%b-%Y') # datetime.datetime(2023, 7, 12, 0, 0) | | |
| **Format Codes** | | |
| **Directive** | **Meaning** | **Example** |
| %a | Weekday as locale’s abbreviated name. | Sun, Mon, …, Sat (en\_US) |
| %A | Weekday as locale’s full name. | Sunday, Monday, …, Saturday (en\_US) |
| %w | Weekday as a decimal number, where 0 is Sunday and 6 is Saturday. | 0, 1, …, 6 |
| %d | Day of the month | 01, 02, …, 31 |
| %b | Month as locale’s abbreviated name. | Jan, Feb, …, Dec (en\_US) |
| %B | Month as locale’s full name. | January, February, …, December (en\_US) |
| %m | Month as a zero-padded decimal number. | 01, 02, …, 12 |
| %y | Year without century | 00, 01, …, 99 |
| %Y | Year with century as a decimal number. | 0001, 0002, …, 2013, 2014, …, 9998, 9999 |
| %H | Hour (24-hour clock) | 00, 01, …, 23 |
| %I | Hour (12-hour clock) | 01, 02, …, 12 |
| %p | Locale’s equivalent of either AM or PM. | AM, PM (en\_US) |
| %M | Minute as a zero-padded decimal number. | 00, 01, …, 59 |
| %S | Second as a zero-padded decimal number. | 00, 01, …, 59 |
| %f | Microsecond as a decimal number | 000000, 000001, …, 999999 |
| %z | UTC offset in the form ±HHMM[SS[.ffffff]] (empty string if the object is naive). | (empty), +0000, -0400, +1030, +063415, -030712.345216 |
| %Z | Time zone name | (empty), UTC, GMT |
| %j | Day of the year | 001, 002, …, 366 |
| %U | Week number of the year (Sunday as the first day of the week) | 00, 01, …, 53 |
| %W | Week number of the year (Monday as the first day of the week) | 00, 01, …, 53 |
| %c | Locale’s appropriate date representation. | Tue Aug 16 21:30:00 1988 (en\_US) |
| %x | Locale’s appropriate date representation. | 08/16/88 (None) |
| %X | Locale’s appropriate time representation. | 21:30:00 (en\_US) |
| %% | A literal '%' character. | % |



|  |
| --- |
| **Datetime Library 3** |
| **timedelta()** |
| The timedelta object represents the difference between two dates or times. timedelta can add and subtract days, seconds and microseconds to/from a datetime object. |
| **Example** |
| from datetime import datetime  difference = date\_2 - date\_1  difference # datetime.timedelta(days=173)  difference.days # 173  now = datetime.now() now # datetime.datetime(2022, 7, 23, 21, 25, 2, 341081)  now + timedelta(days=10, seconds=15) # datetime.datetime(2022, 8, 2, 21, 25, 17, 341081)  now = datetime.now()  now # datetime.datetime(2022, 7, 23, 21, 25, 2, 341081)  now - timedelta(days=10, seconds=15) # datetime.datetime(2022, 7, 13, 21, 59, 41, 100883) |



|  |
| --- |
| **Copy Library** |
| Copy Module is a set of functions that are related to copying different elements of a list, objects, arrays, etc. It can be used to create shallow copies as well as deep copies. Assignment statements in Python do not copy objects, they create bindings between a target and an object. A copy is sometimes needed so one can change one copy without changing the other. |
| **Shallow copy** |
| Shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original. |
| **Example** |
| import copy  a = [[1],[2],[3]]  b = *copy.copy*(a) ## this will copy the list a to list b  print( a) #[[1], [2], [3]]  print( b) #[[1], [2], [3]] |
| **Deep copy** |
| A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original. |
| **Example** |
| *import copy*  a = [[1],[2],[3]]  b = *copy.deepcopy*(a) ## this will copy the list a to list b  a[0][0] = 0  a[1] = None  print(a) #[[0], None, [3]]  print(b) #[[1], [2], [3]] |



|  |
| --- |
| **Itertools Library 1** |
| The itertools module is a collection of tools intended to be fast and use memory efficiently when handling iterators (like lists or dictionaries). import itertools |
| **Accumulate()** |
| Makes an iterator that returns the results of a function. itertools.accumulate(iterable[, func]) passing a function is optional but will result in the items being summed. |
| **Example** |
| data = [1, 2, 3, 4, 5]  result = *itertools.accumulate*(data, operator.mul)  for each in result:  print(each)  # 1 # 2 # 6 #24 #120 |
| **Combinations() / combinations\_with\_replacement()** |
| Takes an iterable and a integer. This will create all the unique combination that have r members. itertools.combinations(iterable, r)  The with replacement allows individual elements to be repeated more than once. itertools.combinations\_with\_replacement(iterable, r) |
| **Example** |
| shapes = ['circle', 'triangle', 'square',]  result = *itertools.combinations*(shapes, 2)  for each in result:  print(each) # ('circle', 'triangle') # ('circle', 'square') # ('triangle', 'square')  shapes = ['circle', 'triangle', 'square']  result = *itertools.combinations\_with\_replacement*(shapes, 2)  for each in result:  print(each)  # ('circle', 'circle') # ('circle', 'triangle') # ('circle', 'square') # ('triangle', 'triangle') # ('triangle', 'square') # ('square', 'square') |
| **count()** |
| Makes an iterator that returns evenly spaced values starting with number start. itertools.count(start=0, step=1) |
| **Example** |
| for i in *itertools.count*(10,3):  print(i)  if i > 20:  break  # 10 # 13 # 16 # 19 # 22 |
| **Cycle()** |
| This function cycles through an iterator endlessly. itertools.cycle(iterable) |
| **Example** |
| colors = ['red', 'orange', 'yellow', 'green', 'blue', 'violet']  for color in *itertools.cycle*(colors):  print(color)  # red # orange # yellow # green # blue # violet # red # orange #..... |



|  |
| --- |
| **Itertools Library 2** |
| **Chain()** |
| Take a series of iterables and return them as one long iterable. itertools.chain(\*iterables) |
| **Example** |
| colors = ['red', 'orange', 'yellow', 'green', 'blue']  shapes = ['circle', 'triangle', 'square', 'pentagon']  result *= itertools.chain*(colors, shapes)  for each in result:  print(each)  # red # orange # yellow # green # blue # circle # triangle # square # pentagon |
| **Compress()** |
| Filters one iterable with another. itertools.compress(data, selectors) |
| **Example** |
| shapes = ['circle', 'triangle', 'square', 'pentagon']  selections = [True, False, True, False]  result = *itertools.compress*(shapes, selections)  for each in result:  print(each)  # circle # square |
| **Dropwhile()** |
| Make an iterator that drops elements from the iterable as long as the predicate is true; afterwards, returns every element. itertools.dropwhile(predicate, iterable) |
| **Example** |
| data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1]  result *= itertools.dropwhile*(lambda x: x<5, data)  for each in result:  print(each)  # 5 # 6 # 7 # 8 # 9 # 10 # 1 |
| **Filterfalse()** |
| Makes an iterator that filters elements from iterable returning only those for which the predicate is False. itertools.filterfalse(predicate, iterable) |
| **Example** |
| data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1]  result = *itertools.filterfalse*(lambda x: x<5, data)  for each in result:  print(each)  # 5 # 6 # 7 # 8 # 9 # 10 |
| **Islice()** |
| This function is very much like slices. This allows you to cut out a piece of an iterable. *itertools.islice*(iterable, start, stop[, step]) |
| **Example** |
| colors = ['red', 'orange', 'yellow', 'green', 'blue',]  few\_colors *= itertools.islice*(colors, 2)  for each in few\_colors:  print(each)  # red # orange |



|  |
| --- |
| **Itertools Library 3** |
| **Groupby()** |
| This function groups things together. itertools.groupby(iterable, key=None) |
| **Example** |
| robots = [  {"name": "blaster", "faction": "autobot"},  {"name": "galvatron", "faction": "decepticon"},  {"name": "jazz", "faction": "autobot"},  {"name": "metroplex", "faction": "autobot"},  {"name": "megatron", "faction": "decepticon"},  {"name": "starcream", "faction": "decepticon"},  ]  for key, group in *itertools.groupby*(robots, key=lambda x: x['faction']):  print(key)  print(list(group))  # autoboot  # [{'name': 'blaster', 'faction': 'autobot'}]  # decepticon  # [{'name': 'galvatron', 'faction': 'decepticon'}]  # autoboot # [{'name': 'jazz', 'faction': 'autobot'}, {'name': 'metroplex', 'faction': 'autobot'}]  # decepticon  # [{'name': 'megatron', 'faction': 'decepticon'}, {'name': 'starcream', 'faction': 'decepticon'}] |
| **Permutations()** |
| Creates iterables of all possible permutations. itertools.permutations(iterable, r=None) |
| **Example** |
| alpha\_data = ['a', 'b', 'c']  result = *itertools.permutations*(alpha\_data)  for each in result:  print(each)  # ('a', 'b', 'c') # ('a', 'c', 'b') # ('b', 'a', 'c') # ('b', 'c', 'a') # ('c', 'a', 'b') # ('c', 'b', 'a') |
| **Product()** |
| Creates the cartesian products from a series of iterables. |
| **Example** |
| num\_data = [1, 2, 3]  alpha\_data = ['a', 'b', 'c']  result *= itertools.product*(num\_data, alpha\_data)  for each in result:  print(each)  # (1, 'a') # (1, 'b') # (1, 'c') # (2, 'a') # (2, 'b') # (2, 'c') # (3, 'a') # (3, 'b') # (3, 'c') |
| **Repeat()** |
| This function will repeat an object over and over again. Unless, there is a times argument. itertools.repeat(object[, times]) |
| **Example** |
| for i *in itertools.repeat*("spam", 3):  print(i)  # spam # spam # spam |



|  |
| --- |
| **Itertools Library 4** |
| **Starmap()** |
| Makes an iterator that computes the function using arguments obtained from the iterable. itertools.starmap(function, iterable) |
| **Example** |
| data = [(2, 6), (8, 4), (7, 3)]  result = *itertools.starmap*(operator.mul, data)  for each in result:  print(each)  # 12 # 32 # 21 |
| **Takewhile()** |
| The opposite of dropwhile(). Makes an iterator and returns elements from the iterable as long as the predicate is true.  itertools.takewhile(predicate, iterable) |
| **Example** |
| data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 1]  result = *itertools.takewhile*(lambda x: x<5, data)  for each in result:  print(each)  # 1 # 2 # 3 # 4 |
| **Tee()** |
| Return n independent iterators from a single iterable.  itertools.tee(iterable, n=2) |
| **Example** |
| colors = ['red', 'orange', 'yellow', 'green', 'blue']  alpha\_colors, beta\_colors *= itertools.tee*(colors)  for each in alpha\_colors:  print(each)  # red # orange # yellow # green # blue  for each in beta\_colors:  print(each)  # red # orange # yellow # green # blue |
| **Zip\_longest()** |
| Makes an iterator that aggregates elements from each of the iterables. If the iterables are of uneven length, missing values are filled-in with fillvalue. Iteration continues until the longest iterable is exhausted.  itertools.zip\_longest(\*iterables, fillvalue=None) |
| **Example** |
| colors = ['red', 'orange', 'yellow', 'green', 'blue',]  data = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10,]  for each in itertools.zip\_longest(colors, data, fillvalue=None):  print(each)  # ('red', 1) # ('orange', 2) # ('yellow', 3) # ('green', 4) # ('blue', 5) # (None, 6) # (None, 7) # (None, 8)  # (None, 9) # (None, 10) |



|  |
| --- |
| **Os Library** |
| Os provides a portable way of using operating system dependent functionality. It works for both windows and unix based (linux / mac ect) (main difference windows uses \ in its paths and unix / )  An absolute path, which always begins with the root folder  A relative path, which is relative to the program’s current working directory |
| **path.join** |
| Can be used to join paths using the format of the operating system. |
| **Example** |
| os*.path.join*('usr', 'bin', 'spam') # 'usr\\bin\\spam' |
| **Current working directory getcwd()/ chdir()/ relpath()** |
| To find the current working directory getcwd() can be used which returns the absolute path. To change the current working directory chdir() can be used.  To get the relative path relpath() can be used |
| **Example** |
| os.*getcwd()* # 'C:\\Python34'  os.c*hdir*('C:\\Windows\\System32')  os.*getcwd()* # 'C:\\Windows\\System32'  os.path.relpath('/etc/passwd', '/') # 'etc/passwd' |
| **Creating a new folder makedirs()** |
| To create a new folder makedirs() can be used to which the path can be supplied. |
| **Example** |
| os.makedirs('C:\\delicious\\walnut\\waffles') |
| **Checking path/file validity path.exists()/ path.isfile()/ path.isdir()** |
| To check if a file existis path.exists() is used.  To check if a path references a file path.isfile() can be used.  To check if a path references a folder path.isdir() can be used. |
| **Example** |
| os.path.exists('test/setup.py') # True  os.path.isfile('test/setup.py') # True  os.path.isdir('test/setup.py') # False |
| **File size path.getsize()** |
| To retreieve the file size in bytes path.getsize() can be used. (also works on folders) |
| **Example** |
| os.path.getsize('C:\\Windows\\System32\\calc.exe') # 776192 |
| **Listign directories listdir()** |
| To get a list of all folders and files in a folder listdir() can be used. |
| **Example** |
| os.listdir('C:\\Windows\\System32') # ['0409', '12520437.cpx', '12520850.cpx', … ] |
| **Deleting files/folder unlink()/ rmdir()** |
| To delete a file unlink() can be used.  To delete a folder rmdir() can be used (will only work on empty folders) |
| **Example** |
| Os.unlink(test/setup.py)  Os.rmdir('C:\\Windows\\System32') |



|  |
| --- |
| **Pathlib Library** |
| Pathlib is basicaully a more extensive version of OS.path, focused more on object-oriented programming. (import pathlib) |
| **Path().joinpath()** |
| Can be used to join paths using the format of the operating system. |
| **Example** |
| print(Path('usr').joinpath('bin').*joinpath*('spam')) # 'usr\\bin\\spam' |
| **Current working directory cwd()** |
| To find the current working directory cwd() can be used which returns the absolute path. |
| **Example** |
| print*(Path.cwd())*  *#* 'C:\\Python34' |
| **Creating a new folder makedirs()** |
| To create a new folder mkdir() can be used to which the path can be supplied. |
| **Example** |
| (cwd / 'delicious' / 'walnut' / 'waffles').*mkdir()* |
| **Checking path/file validity is\_absolute/ .exists()/ is\_file()/ is\_dir()** |
| To check if a path is an absolute path path().isabsolute() can be used.  To check if a path exists path().exists() can be used.  To check if a path is a file path().is\_file() can be used.  To check if a path is a folder path().is\_dir() can be used. |
| **Example** |
| Path('/').is\_absolute() # True  Path('setup.py').*exists()* # True  Path('setup.py'*).is\_file()* # True  Path('setup.py').*is\_dir()* # False |
| **File size path.stat()** |
| To retreieve the file size in bytes path().stat() can be used. (also works on folders) |
| **Example** |
| stat = Path('/bin/python3.10').*stat()*  print(stat)  # os.stat\_result(st\_mode=33261, st\_ino=141087, st\_dev=2051, st\_nlink=2, st\_uid=0,  # …  # st\_gid=0, st\_size=10024, st\_atime=1517725562, st\_mtime=1515119809, st\_ctime=1517261276)  print(stat.st\_size) # size in bytes  # 10024 |
| **Listign directories iterdir()** |
| To get a list of all folders and files in a folder path().iterdir() can be used. |
| **Example** |
| For f in path('C:\\Windows\\System32').*iterdir():*  Print(f)  # '0409'  #... |



|  |
| --- |
| **Random Library** |
| The random module is a built-in module that allow us to generate random elements.  Import random |
| **Random()** |
| Random generates a random float between 0 and 1 . |
| **Example** |
| random.*random()* # 0.13436424411240122 |
| **Randint()** |
| Generates a random integer between the start and stop parameters. |
| **Example** |
| random.*randint*(1, 5) # 1 |
| **Choice()** |
| Returns a randomly selected element from an iterable like a list. |
| **Example** |
| random.*choice([*1, 2, 3, 4]) # 2 |
| **Shuffle()** |
| Takes an iterable and shuffles it. |
| **Example** |
| my\_list = [1, 2, 3, 4] random*.shuffle*(my\_list)  print(my\_list) # [1, 4, 3, 2] |
| **Sample()** |
| returns a list with a random selection from an iterable. The number of elements returned is equal to the k parameter: random.sample(iterable, k: int). |
| **Example** |
| random.*sample*([1, 2, 3, 4], 2) # [3, 4] |
| **Uniform()** |
| the uniform method is similar to randint, but return a floating point number. |
| **Example** |
| random.*uniform(*1, 5) # 3.697943322009309 |
| **Seed()** |
| The seed method is used to initialize the random number generator.  Setting the seed to a number will always return the same random number.  The default value of the seed method is the current system time, that is why we always get a different number. |
| **Example** |
| random.*seed(1)*  random.random()  # 0.13436424411240122  random.*seed(1)*  random.random()  # 0.13436424411240122 |



|  |
| --- |
| **Zipfile Library** |
| Provides tools to create, read, write, append, and list a ZIP file.  Import zipfile |
| **Reading ZIP files** |
| Using a with loop you can open a zip folder. using .namelist() you can retrieve the file names + extensions.  Using .getinfo() you can retrieve a files info like size, compression methoded ect:  Using file\_size you can get the file size.  compress\_size you can get the file size while compressed. |
| **Example** |
| *with zipfile.ZipFile('example.zip') as example\_zip*:  print(example\_zip*.namelist())*  spam\_info = example\_zip.*getinfo*('spam.txt')  print(spam\_info*.file\_size*)  print(spam\_info.*compress\_size*)  # ['spam.txt', 'cats/', 'cats/catnames.txt', 'cats/zophie.jpg']  # 13908  # 3828 |
| **Extracting from zipfiles** |
| The extractall() method for ZipFile objects extracts all the files and folders from a ZIP file into the current working directory.  The extract() method for ZipFile objects will extract a single file from the ZIP file.  By giving them an additional argument you can indicate the extraction folder. |
| **Example** |
| with zipfile.ZipFile('example.zip') as example\_zip:  example\_zip.extractall()  with zipfile.ZipFile('example.zip') as example\_zip:  print(example\_zip.extract('spam.txt'))  print(example\_zip.extract('spam.txt', 'C:\\some\\new\\folders')) |
| **Creating/ adding to ZIP files** |
| To add a file to a zip folder you can use the write method. This is done by in the with loop adding a “w” for write (creating new), “A” for appending new files.  If using “r” you can still use things like namelist() to read the files names just not edit the .zip  Using .write(file, compress\_type) you can add files to the zip folder. |
| **Example** |
| with *zipfile.ZipFile('new.zip', 'w') as new\_zip:*  new\_*zip.write*('spam.txt', compress\_type=zipfile.ZIP\_DEFLATED) |
| **Retrieving all metadata** |
| To retrieve all metadata of a .zip you can use infolist() this will let you print anyinfrom the comment, modified data, system, zip version ect. |
| **Example** |
| with zipfile.ZipFile(archive\_name) as zf:  for info in zf.*infolist()*  print(f'System : {system}') # System : Unix  print(f'Comment : {info.comment}') # Comment : b''  print(f'Modified : {modified}') # Modified : 2022-11-15 06:48:02 |



|  |
| --- |
| **Shelve Library** |
| A “shelf” is a persistent, dictionary-like object. The difference is that the values (not the keys!) in a shelf can be essentially arbitrary Python objects (anything that the pickle module can handle).  Import shelve |
| **Adding to a shelve** |
| First you open a shelf and then you can add key value pairs to it |
| **Example** |
| wife = ['Pretty', 'Lovely', 'Nice']  with *shelve.open*('mydata') as shelf\_file:  shelf\_file['wife\_list'] = wife |
| **Reading a shelve** |
| First open the shelve and then you can access it. |
| **Example** |
| with shelve.open('mydata') as shelf\_file:  print(type(shelf\_file))  print(shelf\_file['wife'])  # <class 'shelve.DbfilenameShelf'>  # ['Pretty', 'Lovely', 'Nice'] |
| **Accessing all keys / values in a shelve** |
| Like a dictonary you can retrieve all keys and values by using .keys() and .values() |
| **Example** |
| with shelve.open('mydata') as shelf\_file:  print(list(shelf\_file.keys()))  print(list(shelf\_file.values()))  # ['wife']  # [['Pretty', 'Lovely', 'Nice']] |